302 – Redis

NoSQL

WIK-NOSQL-302

Intervenant : Jeremy Trufier <jeremy@wikodit.fr>

ynov

wikodit

Au préalable

Redis

- Base clé / valeurs
- Grand nombre de clients sous plus de 50 langages différents
- OpenSource (BSD)
- Stockage en mémoire vive
- Mécanismes de clustering et de haute disponibilité
- Support de scripts LUA

Persistence sur disque

- RDB : Ram snapshot selon intervalles
- AOF : Log les opérations d'écriture
- Grands nombres d'options disponibles

Clustering

- Redis Sentinel : Replication Master/Slave
- Redis Cluster : Sharding (v2+) + Réplication (v3+)

Les clés

- Théoriquement 512MB max
- Recommandation : < 1KB (1000 caractères)
- Construire une nomenclature !

```
user:1052:friends
U:1052/F
U1052F
```
} U = user, F = friends

Les valeurs

- 512MB max
- Types :
  - String / Binary
  - List
  - Hash
  - Sets
  - Sorted Sets
  - HyperLogLogs

Utilisation

Téléchargement

- Windows port : https://github.com/MSOpenTech/redis/releases
- OSX : Homebrew + brew install redis
- Linux : gestionnaire de package ou http://redis.io/download
- **http://try.redis.io/**

Redis CLI

Lancement

Lancer le serveur (terminal 1)

```
$ redis-server
...
```

Lancer et tester la cli (terminal 2)

Création

Récupération

Existence ?

Renommage de clé

Suppression

```
$ redis-cli
> SET mykey toto
OK
> GET mykey
"toto"
> EXISTS mykey
1
> RENAME mykey test
1
> EXISTS mykey
0
> GET test
"toto"
> DEL test
1
> GET test
(nil)
```

INCR

Incrémente la valeur d'une clé, tout en conservant l'atomicité

Par défaut si la clé n'existe pas, la valeur est définie à 0

```
> INCR counter
1
> INCR counter
2
> INCR counter
3
> GET counter
"3"
```

EXPIRE & TTL

```
> SET session:21345:userId "3d29a729-5311-42a5-bffb-af5cdcf56b99"
OK
> EXPIRE session:21345:userId 30
1
> TTL session:21345:userId
12
> GET session:21345:userId
"3d29a729-5311-42a5-bffb-af5cdcf56b99"
> GET session:21345:userId
(nil)
```

EXPIRE	Permet d'ajouter un time-to-live à une clé
TTL	Lit un TTL
EXPIREAT	Expiration à un timestamp Linux
PEXPIRE, PEXPIREAT	Milli-secondes au lieu de secondes

Les listes ordonnées

Ajout à droite

Ajout à gauche

POP à droite
(LPOP = à gauche)
Récupération de parties
du tableau
-1 = éléments à partir de la
droite

Récupération d'une valeur
à un index

```
> RPUSH user:23105:logs "GET /todos" "GET /todos/add"
2
> RPUSH user:23105:logs "POST /todos" "GET /todos/153"
4
> LPUSH user:23105:logs "GET /sessions"
5
> RPOP user:23105:logs
"GET /todos/153"
> LRANGE user:23105:logs 0 -1
1) "GET /sessions"
2) "GET /todos"
3) "GET /todos/add"
4) "POST /todos"
> LRANGE user:23105:logs 0 0
1) "GET /sessions"
> LINDEX user:23105:logs -1
"POST /todos"
```

Les sets

Liste d'éléments uniques non ordonnée

Ajout d'un élément

Suppression d'un élément

Récupération

Vérification si dans le set

```
> SADD status "draft" "published" "awaiting"
3
> SADD status "validating"
1
> SREM status "awaiting"
1
> SMEMBERS status
1) "published"
2) "draft"
3) "validating"
> SISMEMBER status "published"
1
```

Les hashes

Des structures clés/valeurs

Set clé/valeur

Set multiple

Incrémentation

Récupération du hash sous forme de liste ordonnée

Récupération d'une valeur précice

```
> HSET user:21031 pseudo "Tronix117"
1
> HMSET user:21031 firstname "Jeremy" callCount 1
OK
> HINCRBY user:21031 callCount 2
3
> HGETALL user:21031
1) "pseudo"
2) "Tronix117"
3) "firstname"
4) "Jeremy"
5) "callCount"
6) 3
> HGET user:21031 callCount
"3"
```

Collections

```
> HMSET user:21031 pseudo "Tronix117" firstname "Jeremy"
OK
> HMSET user:21032 pseudo "Fitz" firstname "Chivalry"
OK
```

```
> SADD users "21032"
1
> SADD users "21031"
1
```

Slow

```
> KEYS user:*
1) "user:21032"
2) "user:21031"
```

```
> SMEMBERS users
1) "21032"
2) "21031"
```

Fast

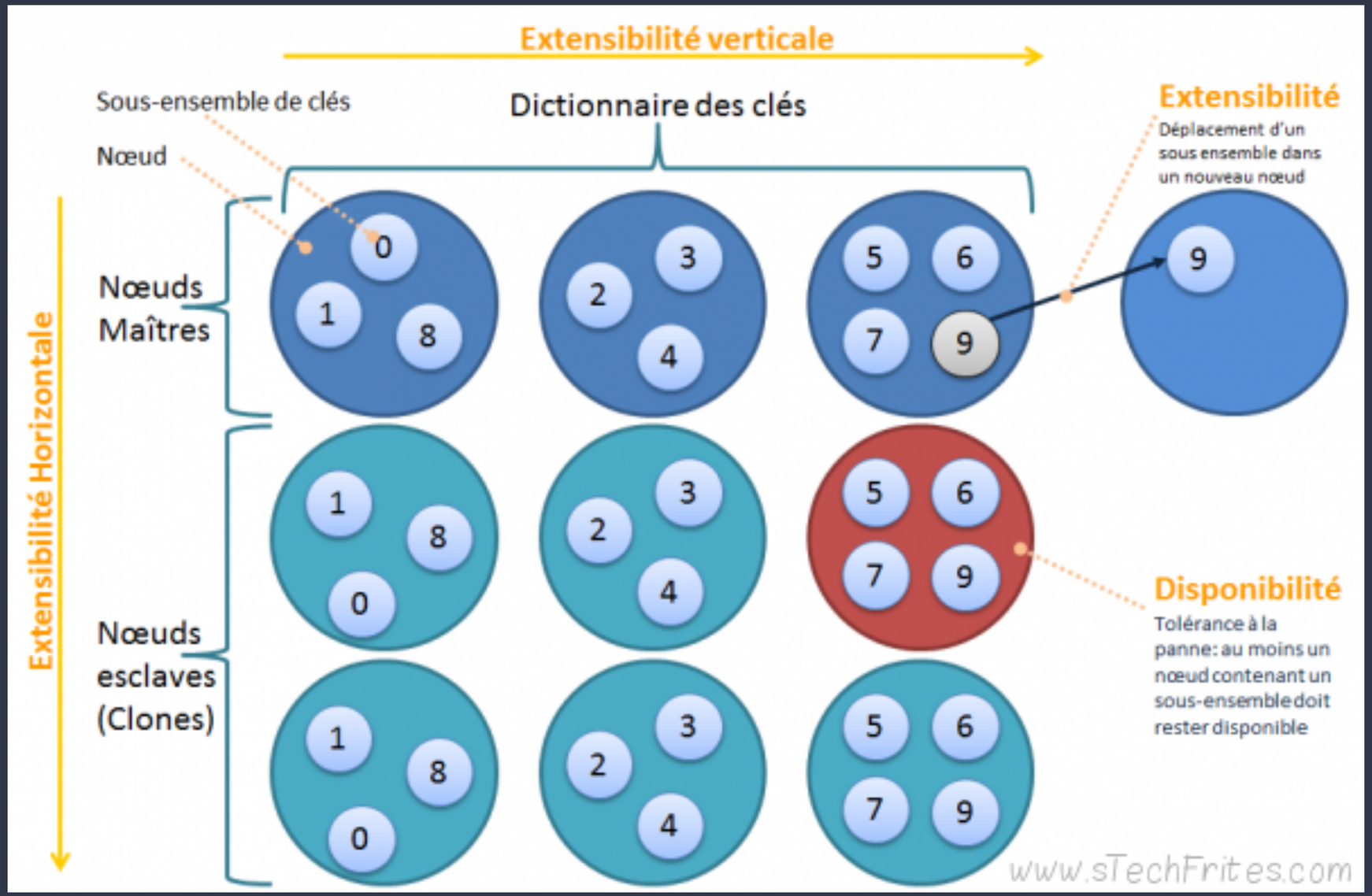
```
> HGET "user:21032" pseudo
"Fitz"
> HGET "user:21031" pseudo
"Tronix117"
```

Réplication

Cluster Redis

- Haute disponibilité
- Replication des données sur Slave
- Répartition des données (Sharding)
- Auto-promulgation d'un slave en master si défaillance

Exemple de cluster



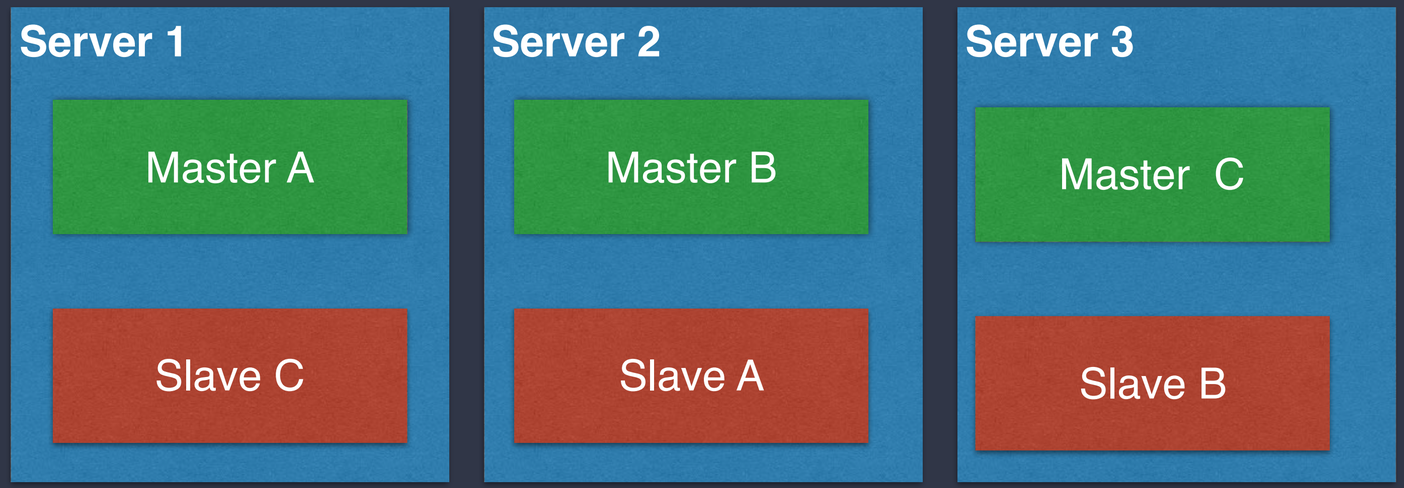
Source image: www.sTechFrites.com

Cluster Redis

- Haute disponibilité
- Replication des données sur Slave
- Répartition des données (Sharding)
- Auto-promulgation d'un slave en master si défaillance

TD : Réplication Redis (réseau)

1. Démarrez 3 machines AWS
2. Démarrez 2 instances de Redis en mode cluster / machine
3. Créez un cluster Redis



4. Ajoutez des données (environ 10)
5. Simuler une défaillance du server 3
6. Affichez le résumé du cluster
7. Affichez les données

Cf. https://redis.io/topics/cluster-tutorial

NodeJS : IORedis

Utilisation

Installation du module

```
$ npm install --save ioredis
```

```
const Redis = require('ioredis')
const redis = new Redis()

async function test() {
  await redis.set('test', 'lol')
  const val = await redis.get('test')
  console.log(val)
}

test().catch((err) => {
  console.log('Error: ', err)
})
```

Accès entier à l'API Redis

Exemple pour enregistrer et lire un Hash en redis

```
const Redis = require('ioredis')
const redis = new Redis()

async function test() {
  await redis.hset('user:1', 'pseudo', 'tronix')
  const val = await redis.hget('user:1', 'pseudo')
  console.log(val)
}

test().catch((err) => {
  console.log('Error: ', err)
})
```

Pipeline

Permet d'envoyer de multiples commandes Redis en une seule fois
(optimise de 50% à 300%)

```
const Redis = require('ioredis')
const redis = new Redis()

async function test() {
  const pipeline = redis.pipeline()
  const userId = require('uuid').v4()

  pipeline.hmset(`user:${userId}`, {
    pseudo: params.pseudo,
    email: params.email
  })
  pipeline.sadd('users', userId)

  await pipeline.exec()
}

test().catch((err) => {
  console.log('Error: ', err)
})
```

Pour plus d'infos sur la bonne manière de gérer des collections en Redis :
http://stackoverflow.com/questions/16375188/redis-strings-vs-redis-hashes-to-represent-json-efficiency

TD : Redis (dev)

Reprendre la correction NodeJS :
https://github.com/Tronix117/wik-njs-303-skeleton
sur la branche qui correspond à votre classe

1. Supprimez les `(\\d+)` dans le fichier `routes/users.js`
2. Editez `models/user.js
3. Supprimer le contenu de chaque fonction dans ce fichier
4. Implémenter ces fonctions de façons à ce que les données soient enregistrée dans Redis
5. Testez avec `npm start`

N'oubliez pas le npm install et npm start!!

Félicitations !!

Cours WIK-NOSQL-302 burned :)